# Proposal cover page

# Transmittal Letter

# A Dynamically Reconfigurable Software Control Architecture for Advanced Life Support

Principal Investigator: Pete Bonasso
CO-Investigator: Dr Cheryl Martin
CO-Investigator: Dr David Kortenkamp

# A Dynamically Reconfigurable Software Control Architecture for Advanced Life Support

## Statement of Justification

The objective of this proposal is to design, develop and demonstrate an intelligent control software architecture for advanced life support (ALS) applications that is dynamically reconfigurable. Such an architecture will dramatically decrease the downtime of ALS systems due to subcomponent failure, and will improve safety and operational efficiency by supporting reconfiguration of the control system with a minimum of deactivation or shutdown of the systems being controlled.

This proposal is relevant to the advanced environmental monitoring control (AEMC) portion of the Advanced Human Support Technology (AHST) Program along several dimensions. Critical to long-duration, crewed missions is the need for a monitoring and control system that can respond rapidly to environmental changes and perform to requirements continuously over a period of years with minimal maintenance (NRC, 1997). This proposal seeks to use an existing architecture, known as 3T, that today meets this stated critical need. Since 1995 we have used 3T to provide autonomous 24/7 controls to several long-duration life-support systems in ground tests at JSC, which demonstrated 3T's ability to provide a dramatic reduction in crew time spent managing ALS systems, with a commensurate increase in safety and reliability (Bonasso, 2001), as well as its ability to provide an integrating strategy among interacting ALS subsystems (Schreckenghost et al., 1998a).

The work in this proposal will revamp and augment 3T to address the specific requirement of the AEMC portion of NRA-02-OBPR-01 that an ALS intelligent control system autonomously assess the AEMC system status and respond, as necessary, to changes in that status (Office of Biological and Physical Research, 2002). We propose to extend 3T to be dynamically reconfigurable whenever a component fails, be it the ALS hardware or the monitoring and control software. That is, 3T will detect the failure, automatically switch to backup hardware or use alternate software if available, thus increasing system reliability, or it will safely bring down only the affected components until they are repaired, thus increasing crew safety through graceful degradation of the ALS. Once the original components are restored, the architecture will dynamically locate them, integrate them into the running system, and resume nominal operations.

In addition to increasing system reliability and safety in deployed ALS systems, a dynamically configurable 3T will also support ALS technology advancement by allowing the rapid integration of newly developed sensors, actuators and/or control algorithms for evaluation or comparison purposes in ALS ground testbeds.

In support of these causes, the software control architecture resulting from this proposal will:

1) Be completely distributed, thus having no single point of failure
2) Provide dynamic reconfiguration of its functions as a result of hardware or software failures with minimal or no requirement to shutdown the control system
3) Provide interfaces that support multiple programming languages
4) Accommodate OPC (OLE for Process Control) sensor and actuator processing
5) Allow dynamic connection of new/alternate software modules with minimum or no requirement to shutdown the control system
6) Operate on any major computing platform in any geographical location reachable by high-speed networks

# Table of Contents

# A Dynamically Reconfigurable Software Control Architecture for Advanced Life Support

## 1 Project Description

### 1.1 Motivation

Advanced life support (ALS) provides basic life-sustaining functions inside spacecraft or on planetary surfaces. ALS includes controlling pressure, temperature and humidity, providing usable water and breathable air; supplying acceptable food; and managing wastes. Critical to long-duration, crewed missions is the need for a monitoring and control system that can respond rapidly to environmental changes and perform to requirements continuously over a period of years with minimal maintenance (NRC, 1997). Over the past seven years we have been providing autonomous, round-the-clock intelligent control for a number of long duration ALS applications using a software control organization known as 3T that today meets this stated critical need. As a particular realization of a more general intelligent control paradigm called the three-layer architecture, 3T has been successful in demonstrating the ability to provide a dramatic reduction in crew time spent managing ALS systems, with a commensurate increase in safety and reliability (Bonasso, 2001), as well as an ability to provide an integrating strategy among interacting ALS subsystems (Schreckenghost et al., 1998a).

But a system to support life in space or on remote planetary surfaces is both highly complex and critical to crew safety. It is therefore essential to have an efficient communications network and an intelligent control system capable of autonomous assessment of monitoring and control system status, and having the ability to respond, as necessary, to changes in that status (Office of Biological and Physical Research, 2002). Such changes include the failure of ALS hardware and/or control software failure due to loss of computers or network resources. Changes in the status of an advanced environmental monitoring and control (AEMC) system occur even more frequently in ALS ground test environments. Ground test engineers must be allowed not only to put a developing ALS system through failure scenarios, but also to investigate advanced technology by substituting new sensor, actuator, or control algorithms into the ground test environment. Moreover, future intelligent control approaches for ALS must also take into account the fact that these systems will be distributed geographically, executing on different software operating systems in different programming languages across a variety of computing hardware.

Given these dimensions of distribution and the need to enable replacement or revision of any existing component for repair or testing, an AEMC system should be designed to detect changes in system status and then to intelligently reconfigure itself such that ALS safety and reliability is maintained. In particular, the AEMC system should

1) Detect loss of hardware and/or software or respond to a user requirement for replacement of hardware and/or software,
2) Automatically switch to backup hardware or use alternate software, thus increasing system reliability
3) If backup hardware or software is not available, safely bring down only the affected components until they are repaired, thus increasing crew safety through graceful degradation
4) Once the original components are restored, locate them, dynamically integrate them into the running system, and resume nominal operations

The 3T system, as successful as it has been, has only a minimal capability to be distributed or to be reconfigured without impact to the ALS system being controlled. However, recent advances in the field of distributed computing can support distributing control systems and can allow them to be dynamically reconfigured to an extent never before realized. Current tools for building distributed and cross-platform systems exhibit not only excellent support for state-of-the-art distributed design methodology but also improved standardization for interoperability across diverse components. More importantly, these tools provide the means to develop capabilities that allow the components of an architecture, like 3T, to be changed *while the system is running*. Thus, in order to realize our goal of a dynamic, reconfigurable 3T system, we propose to (1) update the implementation of the 3T architecture using the recently emerged standards and tools for interfacing distributed software and hardware and (2) use this updated implementation to develop a design to support dynamic reconfiguration of 3T and to explore several research challenges associated with distributed, reconfigurable control systems.

With this program we will bring the power of new distributed systems methodology and the flexibility of dynamic reconfiguration to the already formidable three-layer control paradigm, thereby dramatically increasing the safety, reliability and flexibility of future ALS systems.

## 1.2 Background.

In support of the Crew and Thermal Systems Division (CTSD) of NASA's Johnson Space Center (JSC), we have successfully developed autonomous, intelligent control systems for ALS applications since 1995 (Bonasso, 2001; Lai-fook and Ambrose, 1997; Schreckenghost et al., 1998b). In these projects we have used the three-layer approach to organizing and developing the control software.

### 1.2.1 Three-layer Approach.

The three-layer architecture, developed by the artificial intelligence (AI) community in several forms since the late eighties (a useful historical summary can be found in (Gat, 1998)), separates the general intelligent control problem into three interacting pieces (see Figure 1):

- A reactive bottom layer consisting of a set of hardware specific

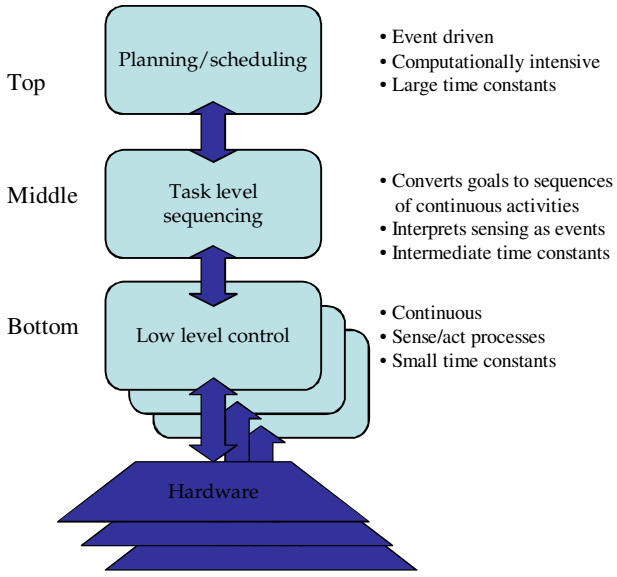



**Figure 1. Three-layer Intelligent Control Architecture**

modules, for example, switchers, low level control algorithms, or data analyzers, which are tightly bound to the specific hardware and must interact with that hardware in real-time.

- A sequencing middle layer, which differentially activates the reactive modules in the bottom layer in order to direct changes in the state of the world and accomplish specific tasks, for example, moving a reverse osmosis (RO) system through its phases of operation.
- A planning/scheduling top layer, which reasons in depth about goals, resources and the future effects of current actions.

The key aspect of the three-layer approach is that it gives developers the ability to integrate the continuous, near-real time control algorithms in the bottom layer with advanced AI algorithms in the top layer — i.e., automated planners and schedulers — which are event driven but more computationally expensive. The architecture achieves this integration through the action of the middle layer. Essentially, the middle layer translates the goal states computed by a planning/scheduling system into a sequence of continuous activities carried out by the lowest layer, and interprets sensor information from the bottom layer as events of interest to the top layer.

### 1.2.2 3T.

Figure 2 shows our particular implementation of the three-layer approach, known as 3T (Bonasso et al., 1997), used in ALS projects to date. Our planning system develops a long-range plan for the ALS operation. Over time, it places tasks on the agenda of the sequencer for execution and monitors the results. The sequencer's plan interpreter decomposes the task into an ordered list of primitives that can be executed by the bottom layer, which is called the skills layer. A "skill" is a robust piece of code for carrying out action or for interpreting sensor information as events. The skills are normally organized into networked groups for each ALS subsystem, for example the air evaporation subsystem (AES) in a water recovery system. For each skill network we use a device skill (d-skill), to manage the low-level data conversion to and from the hardware. The sequencer activates and deactivates skills through a skill manager for each network. As action skills are enabled, event skills are activated to watch for sensor information signaling the completion (or failure) of the action. As each primitive succeeds, the old skills are deactivated and new actions and events for the next primitive are activated. When tasks complete (or fail), the sequencer informs the planner which will update the plan (or replan if necessary) and send new tasks to the sequencer for execution.

Since 1995, we have proven that 3T is a powerful organizing and development principle for the intelligent control of a contained group of life support subsystems, and it also provides for increased safety and reliability of ALS systems. During a 15-day Human Rated Test (HRT), in 1995, we used 3T to monitor and "flight-follow" the control of the environment of a wheat crop chamber providing oxygen to a single crewmember (Lai-fook and Ambrose, 1997). In 1997, in support of a 91-day four person HRT, we developed the software to provide 24/7 control of the transfer of product gases among an air revitalization system (ARS), a plant chamber, and a solid waste incineration system (Schreckenghost et al., 1998a). This test demonstrated 3T's ability to provide an integrating strategy among interacting ALS subsystems. In 1999, we used the architecture to incrementally provide controls to an evolving group of water recovery subsystems, culminating in 2001 in a yearlong integrated test of a biological water processor, an inorganic removal system, a brine processing system and a post-processing system (Bonasso, 2001). During this test, for over 98% of the time, the control architecture operated 24/7, unattended, thus providing a dramatic decrease in user time spent managing ALS systems, with a commensurate increase in safety and reliability.

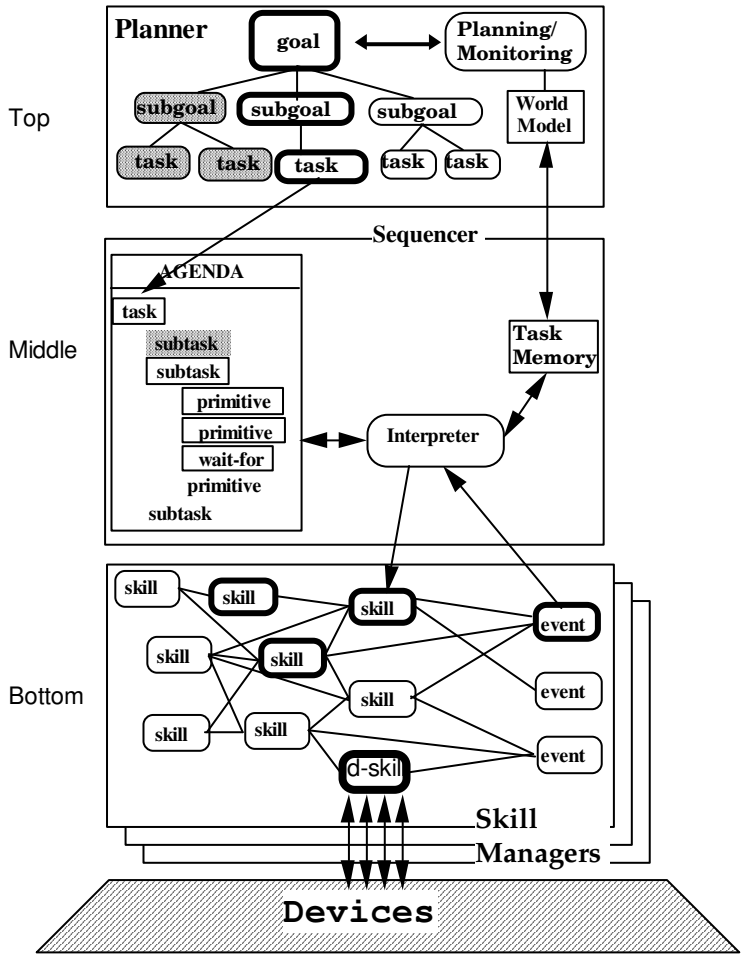



**Figure 2. The 3T Realization of the Three-Layer Control Architecture**

We are not aware of any other ALS control approach that has both the ability to integrate continuous control and state-based planning and scheduling as well as a proven track record of ALS application development as does 3T. And yet, while the benefits of 3T are many, there are key shortcomings in the current 3T implementation that hinder its potential to be dynamically reconfigurable. In particular, the interfaces among the layers in 3T have shown they are limited in their ability to accommodate changes in hardware and software configurations without shutting down the affected systems. The following section discusses the deficiencies of the current 3T implementation in detail.

### 1.2.3 Inadequacies of the Current 3T Implementation

Figure 3, notionally depicts the 3T implementation of the three-layer architecture. The top two layers are each represented by a single box and the bottom layer consists of small boxes representing software controlling the sensors and actuators common to a particular ALS subsystem, for example, the biological water processor or the post processor in a water recovery system. Small horizontal bars within a layer represent the Application Programming Interface (API) from one layer to another. The boxes to the far right represent new control software that might be exchanged into the architecture due to failure or for testing purposes. For instance, an

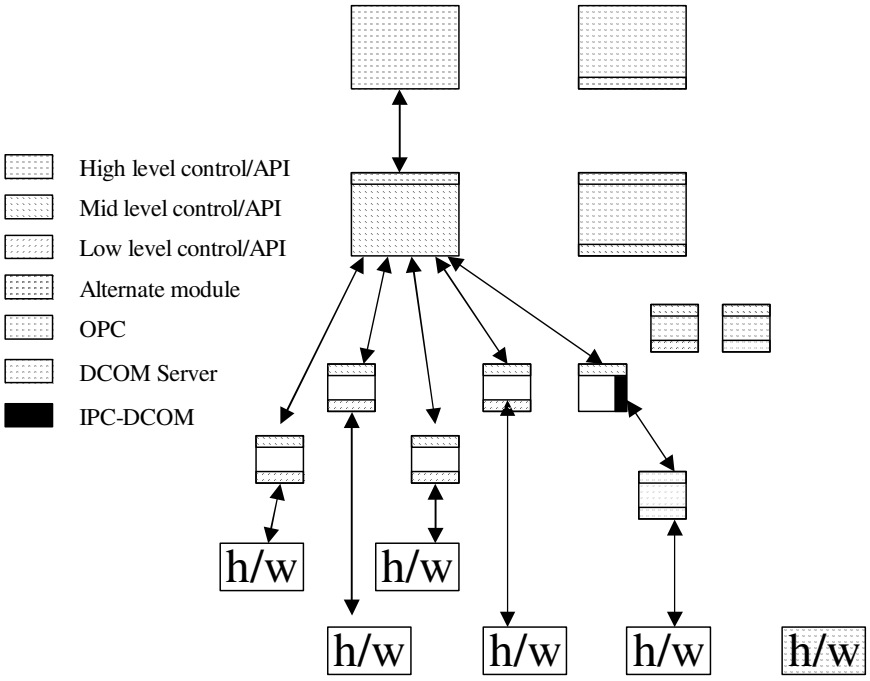



**Figure 3. The various interfaces in the 3T control system. Note that the IPC-DCOM connection is notional since there is to date no implementation of such a bridge.**

alternate scheduler (top tier), control algorithm (bottom tier) or hardware. In the middle tier we might be interested in not only using a replacement sequencer, but also incorporating additional modules, such as a fault diagnosis subsystem that add capabilities to the overall system.

The API for the current implementation of 3T uses a publish-subscribe message passing system known as IPC (Simmons and Dale, 1997)(NDDS is a similar type of middleware; see (RTI, 2002)). To build an interface, the formats of the messages sent from one layer to another must first be defined for both the sending and receiving sides of the interaction. A central server receives and distributes these messages. Then code must be written to construct the messages and/or process the messages (message handlers) on both sides of the interface.

There are two major limiting factors to using any publish-subscribe middleware in distributed software architectures. The first is the fact that a single server accepts and distributes the messages embodied in the interfaces, and thus can be a single point of failure in the system. The second factor is that switching to an alternate software module at any level requires much effort not only to build but also to maintain the resulting system because message construction and processing must be coded and checked for consistency by hand. So making changes to the interfaces dynamically -- i.e., while the software is running -- is simply beyond the capability of publish-subscribe technology, because new messages and handlers cannot be constructed and loaded without recompiling the affected software.

In order to understand both the ramifications of hardware and software changes in the control system, as well as to motivate our specific solution, the next section details the types of changes that ALS systems will be required to accommodate in the near future, and discusses what types of changes are needed in 3T to create a dynamically reconfigurable implementation of the three-layer architecture.

## 1.3 Requirements for ALS Reconfiguration

Changes in hardware and software are the norm in long running, ALS environments. During deployed operations or in a ground test environment, often a sensor or actuator will fail and need to be replaced or an obsolete device will need to be replaced with a more modern version. In addition to these device changes, we foresee the need to dynamically reconfigure future ALS control systems with respect to changes in the control hierarchy and non-hierarchical changes involving software for temporary or periodic use (such as "expert" or diagnostic modules and the user interfaces (UIs)). Each of these types of required changes is explained in the following sub-sections.

### 1.3.1 Hardware Changes and Affected Software.

ALS hardware devices connected to the control system often need to be repaired or replaced. During such procedures, the devices will become unavailable. The control system must tolerate these outages gracefully. In addition, a device's driver software may need to be updated, either alone or in addition to a physical hardware change. In the current 3T implementation, the drivers for each device are compiled directly into the low-level code for that device's entire subsystem (e.g. the subsystem boxes in the bottom layer in Figure 3). So when one driver must be updated for that subsystem, the software for the entire subsystem must be taken offline for the duration of the driver update procedure.

In some cases, a change of device will lead to additional required changes in the control code beyond the drivers (e.g., an entirely new data processing algorithm may be needed to process the temperature from a digital thermometer rather than from a thermocouple). This type of code change also requires the code for the entire subsystem (e.g. the air evaporation subsystem) to be halted, recompiled, and tested. While this is obviously better than bringing the overall ALS system down, we submit that when replacing devices, only the part of the control code affected by those devices (a finer resolution than the subsystem distinction) should be disabled.

Furthermore, even for small changes in device operating characteristics (in the bottom layer), the data the middle layer "sees" will sometimes be different for the new device. What is required to better support the hardware and lower level software changes described above is an interface to and from the hardware to the using software that has the ability to differentially select sets of devices, and that automatically maintains consistency across the interface.

Already on the horizon there is some hope for this hardware dilemma. Manufacturers are beginning to produce sensors and actuators that feed data to new kinds of analog-to-digital (A/D) devices, which convert the data to standard formats and transmit them via Ethernet to data servers (e.g., (Iconics, 2002)) Microsoft's Component Object Model (COM) (Microsoft, 2002) and Distributed COM (DCOM) specifications provide the necessary software infrastructure that defines how applications share data under Microsoft operating systems such as Windows NT. The OPC (OLE for Process Control) specification defines a set of standard COM objects, methods, and properties that specifically address requirements for real-time factory automation and process control applications. OPC servers provide a standard interface to the OPC COM objects associated with the A/D modules, allowing OPC client applications to exchange data and control commands in a generic way. Sensors that take advantage of the OPC approach will always provide their data in a standard format to the rest of the control system, regardless of their operating characteristics. So in the case where a digital thermometer replaces a thermocouple, a temperature in degrees C is served by OPC from either sensor. Further, while the sensor is being changed out, only one channel from the server is affected, rather than all the channels as is the case with hand written code at the bottom level of the current implementation of 3T. However,

OPC is not a complete solution and does not apply to upper layers of the 3T architecture, see Section 1.4.4 for a discussion of this point.

### 1.3.2 Software Changes in the 3T Hierarchy

When a software module in the 3T hierarchy (within one of the three layers) fails in a deployed ALS system, due to network or computer failure, a backup module needs to be brought on-line to take over the function of the failed software. To achieve this reconfiguration, the backup module needs to know what other modules in the hierarchy it must interface with. There is already knowledge at each level of 3T to determine the normal control interfaces. The skill manager knows the skills and hardware normally associated with a given ALS subsystem, such as a biological water processor; the sequencer knows what set of skill managers it normally interfaces with in a given application, and the planner knows what sequencers to interface with. In a distributed setting, the backup module will use this knowledge to locate and connect to the modules to be interfaced with.

In a ground test environment, particularly ones involving large numbers of ALS subsystems (e.g., the Integrity initiative (Henninger, 2001)), determining the usefulness of the control software itself is often the objective of the test. For example, a new algorithm claiming to yield better accuracy from the same water quality sensor may be tested in place of the standard algorithm running in the bottom layer of the architecture. Here, when the user directs the changes in software, the replacing module should be able to query the module being replaced as to the other modules the replacing module needs to connect to. As a matter of fact, in a ground test environment, at any given time the assembled modules may be a mix of standard and experimental software. Thus it becomes important to allow new modules or old modules being reinstated to be able to query their counterparts as to the current set of required connections.

### 1.3.3 Software Connections Outside the 3T Hierarchy

In each layer of the 3T model, there may arise a need for the addition of software modules that augment the software in the hierarchy. Two classes of this type of augmenting module are what we term “expert” software and user interface (UI) modules. An “expert” module adds capability to the control system or to the users of a control system, but is often only temporarily used. For example, to help isolate a problem, users may wish to run a fault diagnosis (FD) module that gathers data from various layers of the architecture and provides analysis to the user or writes its findings to a database. After startup, the module needs to be able to connect to the appropriate modules in the architecture, request the necessary data from those modules and perform its analysis. Gathering the necessary data requires that the expert module not only know the location of the modules in the layers of the architecture but also what functions to call on those modules to obtain the data. Then, after its analysis, the expert module must disconnect from the architecture and shutdown.

Another important aspect of the control of a large suite of ALS subsystems is the need to present the users, be they crew or ALS ground test engineers, with a consistent view of the state of the systems. In some instances, users must also be provided with a means to command portions of the ALS systems being controlled. The layers in 3T continually broadcast their data, so, to display data, the user interfaces (UIs) need to locate the modules in the architecture serving the data. As well, UIs may also allow the user to command portions of the architecture, and so like the expert module, UI modules need to know what functions to call on those modules.

Thus, what is required to support these types of connections is an approach that allows expert or UI modules to locate appropriate software modules to connect to and to remotely invoke functions on those modules.

### 1.3.4 Overall Dynamic Reconfiguration Requirements

To minimize the impact on operations and safety, the intelligent control system must allow the hardware and software changes outlined above to be made *while the ALS system is running*. Because deployed ALS systems are critical, hardware or software replacements should not require the ALS control systems to be shut down and then restarted to accommodate the new device or software. Support for dynamic reconfiguration in support of failed components allows the ALS control system to be designed for failure recovery and gracefully degraded modes of operation at any level. Further, dynamic replacement or addition of hardware or software at any of the three levels in a ground-test environment improves operational efficiency and preserves the continuity of the test. The same benefits also derive from the ability to dynamically replace or add user interfaces or expert modules to the control system. Dynamically integrating a fault diagnosis (FD) module into the control system allows the test engineers to assess the system as it exists, without the effort of attempting to recreate the problem after the system is restarted with the FD package installed. Similarly, dynamically integrating UI modules allows users to bring up UIs from locations and machines convenient to them without disturbing other ALS operations.

In addition to hardware and software replacements or additions, computer or network failures may require a dynamic re-hosting of parts or all of the control system on backup computers. Also, the computing resources required for some software modules may not be known until their runtime. Modules may need to be re-hosted to balance computing resource requirements. In addition, a user may want launch a UI to view the skills data on his laptop from his office or crew stateroom. Therefore dynamic reconfiguration also implies that the control system needs to be able to execute any of its modules on alternate machines or computing platforms.

Another important aspect of the reconfiguration problem is that it should support the integration of the widest possible variety of heterogeneous software modules. For example, the three-layer approach to intelligent control was motivated by the need to use AI techniques for the "intelligence" in intelligent control applications. As such, the top two layers tend to have AI algorithms — many written in languages such as Lisp — while the bottom layer can have control code written in C, C++, Java, or a specialized hardware programming language such as LabView (National Instruments, 2002). The point is that a distributed interface approach for dynamic reconfiguration of an ALS control system must support cross language compatibility.

In summary, to realize dynamic reconfiguration due to hardware or software failures or due to user-driven needs for hardware or software changes, we require a more powerful distributed approach for implementing the 3T control system. Such an approach should

1) not rely on a single server, thus eliminating a single point of failure;
2) accommodate a wide variety of heterogeneous components (e.g., components written in multiple languages as well as both OPC and non-OPC sensor and actuator processing);
3) provide a means for a module on one side of an interface to locate modules on the other side and ascertain and invoke the functions of those modules;
4) operate on any computing platform in any geographical location reachable by high-speed networks; and
5) not require the ALS control systems to be shut down and then restarted to accommodate a new device or software module.

The following sections describe a solution that meets all of these requirements.

## 1.4 Tools For a Reconfigurable 3T

The approach for creating a reconfigurable 3T relies on defining a common set of software interfaces (APIs) that allows any combination of control modules to be interchangeable and that provide the mechanisms for dynamic reconfiguration. These interfaces must be easily specified in a standard manner and must allow a straightforward incorporation of code changes on either side of a given interface.

### 1.4.1 CORBA-based Distributed Systems

We propose using the Common Object Request Broker Architecture (CORBA) (OMG, 2002) to define and implement these interfaces. CORBA provides and/or supports each of the elements required for our reconfigurable 3T approach. CORBA has been used extensively in a number of distributed processing applications, providing an open infrastructure for computer applications to work together over Ethernet networks. CORBA is independent of both operating systems and programming languages, and tools for developing interoperable CORBA applications are available from many different vendors. Recasting the 3T control paradigm in the CORBA framework will allow control components that implement the same interface to become interchangeable with a minimum of recoding, recompilation or hardware downtime.

The investigators for this proposal have first hand experience in developing distributed CORBA applications from the ground up. One such application involved supporting a human watch team for an integrated water processing facility at NASA-JSC (Schreckenghost et al., 2002). In only a few months time we had designed, developed and connected some twenty-six CORBA objects to the output of the water processing control system to watch for shutdown problems in the facility. These objects included a crew activity planner and task management, location and notification services for three human engineers, as well as user interfaces implemented as both graphics and notifications via email. Without CORBA, such an endeavor would have taken two or three times longer to implement. Further, the complete set of object modules, written in Java and Lisp, could be executed simultaneously on different computers in multiple locations using both Windows and Linux operating systems. The following section describes how CORBA works to provide distributed software interoperation.

### 1.4.2 Fundamental CORBA Operation

The definition and use of CORBA-based interfaces relies on the Interface Definition Language (IDL) of the Object Management Group (OMG, the authors of the CORBA specifications) and its associated processing. OMG IDL is used to define APIs to be used by CORBA-enabled processes. The IDL is best described using an example. Figure 4 shows a simple IDL definition that declares the interface between the top-layer planner and any sequencer at the middle layer of the architecture. Although IDL syntax is similar to C++, an IDL file can be compiled by a CORBA tool called an “IDL compiler” into various programming languages (C, C++, Java, Lisp, Smalltalk, ADA, Python, COBOL, etc.).

In the Planner module, shown in Figure 4, there is one interface that defines an object class called ForAllSequencers. The interface defines a method, called reportNewEvent, which can be invoked on an instance of that class. We can see that the method has four arguments: the name of the sequencer, the event identification number and type, and the event data itself. Although the details of the data structures for the method’s arguments are not shown here, they will also be completely specified in the IDL. Essentially the IDL shown in Figure 4 (the term IDL is used for both the interface definition language and an interface written in the language) states that, to report an event, a sequencer must remotely invoke the reportNewEvent method with

```
module Sequencer {

  interface ForPlanner {

    // A method used by any planner to post a new task
    // to the sequencer's agenda.

    void postNewTask( in int task_id,
                      in 3T::taskStructure task);

  };
};
```

**Figure 4. An example interface definition for a sequencer and a planner in 3T**

four arguments of the given type on an instance of the ForAllSequencers object in the planner module.

As shown in this example, the IDL defines an API by specifying the names and arguments of methods. For all major languages, CORBA provides an IDL compiler that automatically generates interface code, i.e., the mechanics of sending and/or receiving. All that remains for the programmer is to implement the "meat" of the function to perform the desired logical operations or carry out the desired algorithm. In this way, using CORBA tools makes development and maintenance of distributed systems much easier than using a publish-subscribe system with static message definitions, which must be maintained and checked manually for consistency. The following paragraphs describe the details of implementing a system with the IDL definition in Figure 4.

A CORBA IDL compiler or "stubber" processes an IDL into two sets of language-specific interface code, one set used by a process sending/making a remote function call across a network (the client) and one set used by a process receiving the remote function call (the server) (see Figure 5). These sets of generated code are then compiled or loaded into the client or server implementation. The generated code in turn interacts with a CORBA backbone implementation called an Object Request Broker (ORB) to make appropriate socket and network calls to achieve remote method invocation (RMI).

For example, if the planner were written in Lisp and the sequencer in C, the developer would execute a Lisp stubber and a C stubber on the same IDL file to produce, respectively, code appropriate for loading into a Lisp image for the planner, and code to be compiled and linked into the sequencer C application. The generated Lisp code provides the implementation "skeleton" necessary for the planner to receive distributed calls to its executable reportNewEvent method,

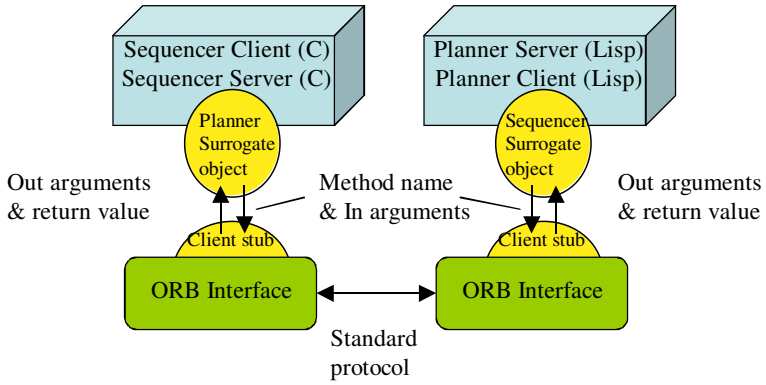



**Figure 5. CORBA processing of remote method invocation (RMI).**

but the Lisp programmer is free to implement the underlying method's actions in any way to perform the associated task.

Similarly, an IDL can be written for the sequencer to allow the planner to give the sequencer new tasks (i.e., calls from the top layer to the middle layer). In this case, the planner is the "client" making the function call, and the sequencer is the "server" receiving the call (see Figure 5).

In summary, CORBA gets its power and flexibility from using language-neutral IDLs as interface specifications, and by the existence of ORB implementations for all major languages. The ease and efficiency of cross-language system integration is therefore greatly improved by using CORBA. Further, a distributed set of CORBA objects does not depend on a single server and thus eliminates the single point of failure of the publish-subscribe approach.

### 1.4.3 CORBA Exceptions, Registries and Dynamic Invocation

Three additional CORBA concepts that are important to the new 3T design discussed later are exception handling, use of registries, and dynamic invocation methodology. Built into the CORBA development tools is a capability for a server to signal that an error occurred — an exception — during a method invocation by a client. Exceptions are delineated by type, and for each exception type, the programmer defines an exception handler that determines how the client should respond to the error. For example, the client can halt operations involving the failed server, or better, can seek out alternate module that has a similar capability as the failed object. Exceptions serve as a mechanism for debugging, but more importantly can be used as run-time indicators that a server object no longer exist or, if it exists, may not be functioning.

To find other objects in a group of distributed modules, CORBA provides a Name Service, which serves as a directory of the names and locations of CORBA objects running in a distributed CORBA application. When each object starts up, it registers its location information with the Name Service, thus allowing other objects to locate and use the services of that object. We will make use of a similar concept termed a registry, which is a module that serves as a database of the locations of active 3T modules with additional information concerning their connectivity status. Modules can update the registry information with their most recent status as well as query the registry for the location and status of other modules.

In some cases a module connecting to another module may need to invoke a method without knowing the details of the interface before runtime. For example, in a test environment, the interface of an experimental software module may be in flux as the designers learn how to better incorporate it into the control system. Rather than generating new interface code for the using module each time, the using module can query the experimental module for its methods, argument lists and result lists at runtime.

With these three developmental concepts in addition to fundamental CORBA operation, we can design and build a 3T implementation that meets all of the dynamic configuration requirements spelled out in the previous section.

### 1.4.4 CORBA Interoperation with DCOM/OPC

A further advantage of a CORBA 3T implementation arises from the fact that CORBA can interoperate with the Distributed Component Object Model (DCOM) and OPC. Recall from Section 1.3.1 that hardware manufacturers are beginning to address some of the interchangeability and reconfiguration problems associated with maintaining systems of sensors and actuators by using DCOM/OPC to standardize data formats and provide distributed access. Through a DCOM/CORBA bridge, our proposed three-layer CORBA implementation can leverage these advances in sensor and actuator technology where they are available.

This DCOM/CORBA bridge will allow 3T to gain the benefits of both CORBA and OPC. Although OPC can solve some of the problems associated with changing out hardware, it does not provide a complete solution. OPC is supported primarily on Windows operating systems, but 3T runs on Unix, Linux, Macintosh and Windows operating systems, which are all supported by CORBA. Further, many ALS devices now in use and planned for use for the next several years do not adhere to the OPC standard. So a requirement still exists to write 3T bottom layer code to interface non-OPC devices to the rest of the control system. Finally, because OPC serves only as an interface to sensors and actuators, it only partially addresses problems associated with interchanging software at the bottom layer of the three-layer architecture, and does not address similar problems in the middle and top layers. OPC's underlying DCOM specification could theoretically be used for the upper 3T layers, but this would not be practical due to limited DCOM language support. As mentioned earlier, many of the AI algorithms in the upper layers of 3T will use Lisp as the programming language. Although CORBA supports Lisp implementations, DCOM does not. Overall a CORBA/DCOM bridge provides the best of both worlds.

## 1.5 A Dynamically Reconfigurable 3T Solution

This section describes our approach to making 3T dynamically reconfigurable using CORBA. We will describe the planned CORBA implementation at each level of 3T as well as how the resulting AEMC system will provide for the hardware and software changes discussed in Section 1.3. Our recasting of 3T into dynamically reconfigurable control architecture consists of first implementing CORBA IDLs for all the 3T interfaces shown in Figure 3 (revisions shown in Figure 6), and then using exception handling, registries at each level, and when needed, dynamic invocation methodology to allow the connecting and reconnecting of hierarchical, and non-hierarchical modules in the architecture.

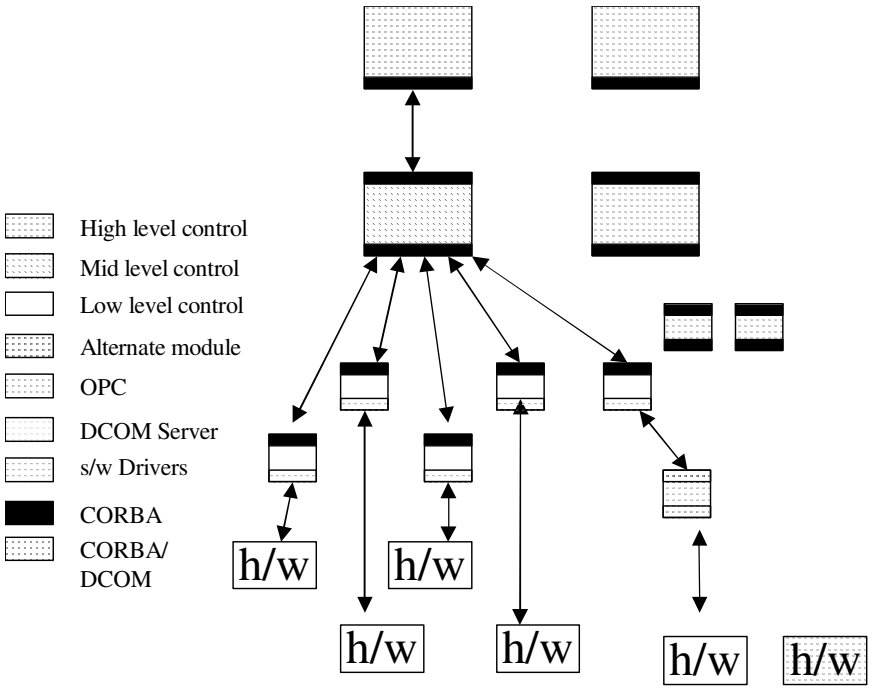



**Figure 6. 3T with interfaces redesigned using CORBA**

### 1.5.1 Implementing 3T Using CORBA

Before starting our investigations into dynamic reconfiguration, we must recast the 3T interfaces in the CORBA framework at each level of the architecture. Figure 6 shows our view of 3T with the interfaces redesigned using CORBA. As is evident by the black interfaces in the figure, CORBA is used throughout the proposed implementation. Wherever a module cannot take direct advantage of CORBA — such as an OPC server — we will design CORBA wrappers so that those modules appear as CORBA objects to the rest of the architecture.

Figure 7 shows a typical skills network for an ALS subsystem (i.e., one block in the bottom 3T layer, for example a post processing system) in 3T. We will design the interfaces among the skills so that sensor data is "pushed" from the device skill to the other skills and actuation commands are pushed to the device skill from the other skills.

As Figure 7 shows, it will be at the device skill that we will use a CORBA-DCOM bridge to provide access to OPC-based sensors and actuators where necessary. The Object Management Group has published a specification for such a bridge (the OMG COM/CORBA Interworking Specification) and several research groups have implemented them (Geraghty et al., 1999). Using CORBA allows us to bring alternate hardware on-line without bringing down the system. For non-OPC devices we plan to explore writing interfaces that breakout the hardware channels in a similar fashion.

In the middle layer of 3T, we will define IDLs for the sequencer that will specify data structures and methods to be used by any skill to send event data to the sequencer. As well, an IDL for each skill will be defined that will allow the sequencer to activate and deactivate that skill (see Figure 7). The existing IPC message formats and associated handlers will serve as a guide for the number and types of interfaces needed in the IDLs. Our IDLs for the sequencer will also include interfaces for applications at the top layer of the architecture such as planners and schedulers. The sequencer not only accepts tasks for its agenda from the top layer of 3T, but also provides execution information from its memory in the form of answers to queries generated by the top layer. Methods for these data exchanges will be included in the sequencer IDL.

The top layer of 3T must invoke methods on the middle layer, through the sequencer IDL defined for the middle layer in the proposed implementation. Applications in the middle layer will call methods in the top layer, via the IDL for the planner.

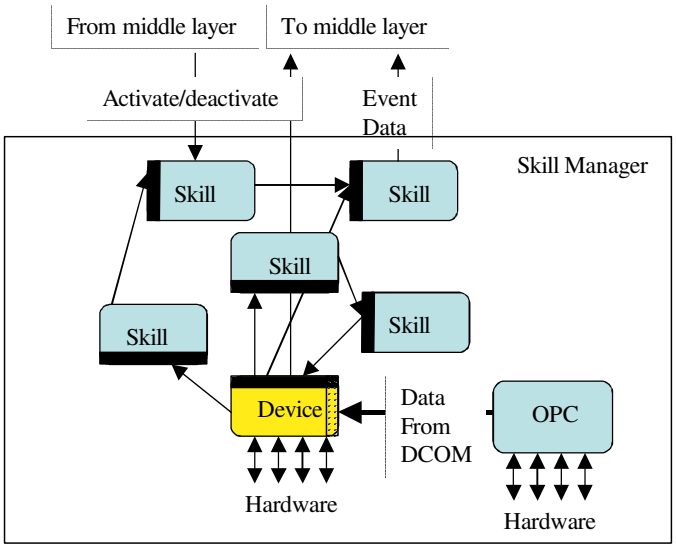



**Figure 7 . Notional 3T Skills Network with Device Skill**

### 1.5.2 Dynamic Reconfiguration

Once we recast the 3T interfaces in the CORBA framework at each level of the architecture we can begin our investigation into dynamic reconfiguration. As discussed earlier, dynamic reconfiguration concerns two aspects of a distributed control system: one, the detection of failures in the hardware or in any part of the architecture and the subsequent reconfiguration for degraded mode operations, and two, the dynamic connecting and reconnecting of alternate modules for test, diagnosis, evaluation or user interface. Therefore this proposal seeks to test the following hypotheses:

**Hypothesis 1:** we can use the CORBA exception handling, and registry and dynamic invocation methodologies to detect inoperative objects in the control system and to reconfigure for degraded mode operations with no system downtime.

**Hypothesis 2:** we can use registry and dynamic invocation methodologies to connect and reconnect hierarchical, user interface and expert modules of a new or temporary nature throughout the ALS 3T control architecture with little or no system downtime.

The key issues we will investigate with regard to the first hypothesis are 1) can the control system detect a module failure, 2) having detected the module failure, can the system reconfigure itself to operate in a degraded mode or, if a degraded mode is not possible, otherwise safe itself, and 3) can the system resume nominal operations once the failure has been rectified. An important issue associated with the second hypothesis is whether the system can automatically coordinate multiple module replacements dynamically without disrupting the rest of the control system. For example, there may be circular run-time dependencies among the modules that cannot be handled solely by exceptions. Our research on this issue will focus on minimizing the amount of downtime. Another key issue associated with both hypotheses is how does the overall control system manage the inertia present from previous operations and commands. That is, either 1) in the case of failure, how can the control system accurately and efficiently resume operations from the point of failure, or 2) in the case of switching to alternate control modules, how can the control system synchronize the new module with the system state given the previous operations.

In the rest of this section we discuss our detailed approach to achieving failure detection and dynamic reconfiguration. We first discuss failure detection, and then dynamic reconfiguration due to failures, followed by dynamic reconfiguration due to a user desire to substitute alternate hierarchical software. Finally we discuss dynamically connecting expert and UI modules. In all cases, we assume that the alternate modules will be CORBA or DCOM-bridge compatible in order to take advantage of the CORBA capabilities described earlier.

**Failure detection.** Two types of failures can occur -- failure of the ALS hardware and failure of a software control module in any layer of the architecture. Hardware failures are detected by the device skills in the bottom layer. Rather than use exceptions, we will leverage an existing 3T capability to flag invalid data. Along with the data passed, each skill in a skill network outputs a flag indicating whether the data is valid. A skill receiving invalid data from another skill, either uses internal default values for its processing or forwards the invalid data flag. Eventually, the skill manager forwards the invalid data flag to the sequencer.

A software module fails because the computer hardware on which it executes fails or there is a failure in the communications network supporting that module. For failure detection in the top and middle levels of the architecture we will make use of CORBA exception handling. When a module in either layer (a client) invokes a method in another module (server) either within the layer or in the other layer, the client module will detect the absence or inactive status of the server module through CORBA's exception handling facilities. In the bottom layer of the

architecture, because each skill is "pushing" its data to the using skills, the receiving skills will not be able to use exception handling to notice the loss of the sending skills. Therefore, in the bottom layer, we will rely on the skill manager and a registry module to keep track of the status of software modules. The registry will be initialized at startup with a list of available skills and their connections to other skills. Periodically the skill manager will conduct a low level poll of all the skills in the network, using exception handling to detect failures. When a skill fails, the skill manager will notify the affected skills and remove the failed skill from the registry. For loss of the sequencer in the middle layer, the skill manager will rely on 3T's existing watchdog skills. A watchdog skill exists in every skill network, notices when communications with the sequencer are lost and commands the ALS hardware to predetermined safe states.

In fact, as we will see, the registry methodology will help at all layers, i.e., we will also have a registry of skill managers at the middle layer and a registry of sequencers in the top layer. Like the skill managers, the sequencer and planner will periodically update the list of locations of active modules in their registries, so that any client module can locate a server at the required level. The managing entity at the appropriate layer will detect registry failures.

**Reconfiguration from Failure.** For hardware failures, we will use some mechanisms that already exist in 3T as well as extensions for operating in degraded mode. At the bottom layer the device skill will notify the using skills via the invalid data flag, the skill manager will notify the sequencer in the same way and both layers will use default values or avoid computations or procedures requiring the given hardware. If the hardware loss is critical, the sequencer will command the affected subsystem to a safe state and notify the planner and the users. Once the hardware is repaired or replaced, the data flags would become valid and the affected modules would resume normal operations. Finally, if backup hardware is online, we will develop algorithms to search those out and bring them to bear in place of the failed hardware. This will be possible because of our breaking out of the channels of the hardware (see Implementing CORBA in 3T above).

Once a software failure is detected, the action taken will be similar for each layer of 3T. When a level detects a failed module, the preferred action will be to have the managing entity for the affected level find another available computing resource and spawn a backup module on that resource (if no resource is available, the default action, currently used in 3T is to operate in a degraded mode by avoiding the use of the failed module). The managing entity for the middle layer is the planning level; the managing entity for the skills is the sequencing level. The managing entities will locate computing resources by querying computer-status modules, activated on each computer available to support the control system at startup. Those modules will also be able to create new modules of any kind specified by a client. A registry module in the middle layer will periodically query the computer-status modules and update databases of available computing resources. If a registry fails, a replacement registry can be bootstrapped from interaction with the existing modules in the system at that layer. As a last resort, if both a registry and it's registered modules fail, the system can be reinitialized at that layer using a default data file of objects for that layer.

The general process for reconfiguration after a software failure is as follows: When a skill fails, the sequencer will first temporarily safe the affected subsystem if necessary. Then it will query the computer-status registry to locate a secondary computing resource. Once this resource is located, the sequencer will spawn a new version of the failed skill, and, using the skill manager registry, inform the new skill of the skills with which it should connect. If a sequencer fails, the planner will take a similar action to spawn a new sequencer. The new sequencer knows the skill sets it must control, e.g., the water recovery sequencer always manages four water-processing subsystems, and will locate the appropriate skill managers via the skill manager registry. The new sequencer will query the skill managers as to the current state of the subsystem. Based on the current situation and the state of the world as obtained from the sequencer, the

planner will generate a new plan and post the new goals to the sequencer. If the planner fails, the affected sequencers will carry out their last assigned tasks and then put their respective subsystems in standby mode as in the current 3T processing. It is then left to the user to start an alternate planner, using information from the computer-status registry. Using module information from the startup initialization file, the new planner will locate and connect to the middle layer sequencers. Then the new planner will accept new goals from the user, generate a plan based on the current situation and the state of the world as obtained from the sequencers, and use the rest of the architecture to execute that plan.

One can see that our proposed answer to the issue of inertia rests in the reasoning power of the planner. AI planners are designed to find a sequence of actions that will bring about a desired state of the world given an initial state. They do so while taking into account resources such as supplies, stores, and time. So when an alternate sequencer or planner comes online, the planner will construct a new plan for the affected ALS subsystems which will bring those systems back online (possibly from safing states), given the existing state of those systems and the ongoing goals provided by the users.

**Reconfiguration for Alternate Modules.** The primary difference between reconfiguring due to failure and reconfiguring because the user wishes to substitute an alternate software module in the hierarchy is the fact that the old module has not failed and so can "hand-off" its control to the new module before shutting down. This hand-off will include informing the new module as to the connections, current state and current activity of the old module. The new module can then inform the modules connected to the old module to disconnect from the old module and reconnect to the new one. Finally, the new module can tell the old module to deactivate.

An important note here is that we consider hierarchical modules that need to be changed and recompiled because their function or their interface has changed to be alternate modules. Thus, the control system need not be stopped to add these modules; but rather, they will be switched in dynamically using the hand-off methodology.

**Connecting UIs and Expert Modules.** In the case where the user desires to connect UIs or expert modules, he simply startups up the module on an available computing resource. Once started, the new module will post its location and active status to the registry at the appropriate level. Based on the module's function, it will know which modules in the hierarchy it must connect to and will locate those modules via the registries at the appropriate level. When connected, the module will invoke the appropriate methods on the connected modules in order to carry out its function, and when that function is completed, will disconnect and remove itself from the registry.

User interface code can exist at any level of the 3T architecture. Currently, we use a standard set of IPC messages to communicate skills, sequencer and planner data to the user. Using these messages and their handlers as guides, we can define IDLs for interfaces at each level of the architecture and these modules can be executed in the same manner as the expert modules. Because there may be a large number of UIs in use in support of the crew or test engineers, we will investigate efficient ways to dynamically register and provide data for multiple UIs. Additionally, as discussed before, some UIs allow the user to command portions of the control system. Those types of UIs will obtain the commanding methods from modules in the hierarchy using dynamic invocation methodology, that is, by requesting the methods and their invocation signature at run-time from the sending modules. The added advantage here is that these UIs can be executed on any networked computer running any operating system supported by CORBA without any previous (compiled-in) knowledge of the methods the UIs may need to invoke on these modules.

### 1.5.3 Experiments in Reconfiguring the 3T Control Architecture.

When the CORBA redesign of 3T is complete, 3T should be a true plug-and-play control architecture, allowing the change out of sensors and actuators, and control and interface modules that will automatically find each other and configure themselves appropriately at any level of the architecture with little or no downtime. To test our hypotheses we will conduct experiments in the new CORBA-based 3T to determine its ability to accommodate ten categories of reconfigurations without bringing down the affected systems. The categories of experiments in dynamic reconfigurations are shown in Figure 8 (again, we assume all new modules are written in the CORBA framework).

1) Device failure or replacement. Invalid data flags will propagate through the skill network, causing the sequencer to either deactivate the affected skills or use online backup hardware if available. Once the device is repaired, valid data flags will signal the sequencer to restore nominal operations.
2) New device. If the device uses the same control algorithm, detection and reconfiguration will be as in 1). If the device requires a new control algorithm, detection and reconfiguration will be as in 4).
3) Failed skill module. The skill manager will detect the failed skill, inform the sequencer, and the sequencer will spawn a new copy of the skill on an alternate computing resource if available. The new skill will use the skill registry to reconnect to the skills network. If no alternate computing resource is found, the sequencer will reconfigure for degraded mode operations as in 1).
4) New low level control algorithm. The user will start up the new skill and inform the sequencer, which will deactivate the old skill and activate the new one. The new skill will use the skill registry to reconnect to the skills network.
5) Failed sequencer. The bottom layer will safe the affected subsystems while the planner spawns a new instance of the sequencer on an alternate computing resource. The new sequencer will locate and connect to its skill networks, determine the current state of the subsystems, and pass that state to the planner, which will generate a new plan and post new goals to the sequencer.
6) Alternate sequencer. The user starts up the alternate sequencer that will use the sequencer registry to find the old sequencer. The old sequencer informs the new sequencer as to its connections, current state and current activity. The new sequencer informs the planner and skill managers connected to the old module to disconnect from the old module and reconnect to the new one then deactivates the old sequencer.
7) Failed planner. The user will bring an alternate planner online. Based on state information obtained from the sequencers the planner will builds and execute a recovery plan.
8) Alternate planner/scheduler. The user will start up the new planner, which will obtain current connections, goals and state information from the old planner. The old planner will disconnect from the middle level and shutdown; the new planner will connect to the middle level and replan ALS activities.
9) Expert application in the top two layers. When started up, the expert module will use the layer registries to locate and connect to the appropriate data sources in the hierarchy. It will also register with the appropriate layer registry. After completing its task, the module will deactivate itself.
10) New user interface. The UI will connect to the hierarchy as in 9), but will also obtain methods for any required commanding from the appropriate hierarchy modules using dynamic invocation methodology.

**Figure 8 Ten Experiments in Dynamic Reconfiguration**

### 1.5.4 Measures of success.

While length of downtime and time to reconfigure are obvious metrics for evaluating 3T's ability to be reconfigured dynamically, other less obvious, less quantifiable measures will also be important to get a complete picture of the value added of the CORBA framework. Such measures include additional lines of code to maintain the system over that used in the current architecture, communications overhead incurred as a result of using remote method invocation, and the overall capability of the control system to maintain control of the ALS system, i.e., the ability of the control system to carry out the explicit mission of the ALS system.

The following is a list of our proposed metrics to be taken for each experiment (where applicable):

1. Number of minutes downtime per affected module
2. Number of seconds for anomaly detection and degraded mode reconfiguration
3. Number of seconds to detect restored module and reconfigure to nominal operations
4. Number of failures to find alternate modules
5. Number of failures to connect to existing modules (UIs and expert modules)
6. Number of communication deadlocks
7. Number of ALS functions lost or failed during reconfiguration
8. Average lines of IDL code required per module
9. Average lines of code required to interface to IDL stubs
10. System loading -- as measured by average response time at each layer of the architecture

## 1.6 Plan for Accomplishing the Work .

To realize the design described in the previous section, we will take a three-phased approach, suggested by the three layers of the architecture. In particular, in each year of the effort we will use CORBA to re-implement a layer starting with the bottom layer, and then carry out our experiments in dynamic reconfiguration on that layer. As we move to the second and third layers, our experiments will include the previous layer so that by year three we will have tested the ten reconfiguration possibilities detailed in the previous section. In all cases we will collect measurements based on the metrics list in the previous section and document our findings,

The application backdrop will be the control of an integrated water recovery system (iWRS), previously developed and run autonomously for two years (Bonasso, 2001). The iWRS consists of four subsystems: a biological water processor (BWP) for the degradation of total organic carbons and the removal of ammonia, a reverse osmosis (RO) system for the removal of inorganic salts, an air evaporation system (AES) to recover water from the brine produced in the RO, and a post processing system (PPS) to remove trace organics and inorganics, resulting in potable water. Simulations in JSC's Intelligent Systems Laboratory (ISL) will support experimenting without iWRS hardware; however, we plan to supplement those experiments with experiments on the PPS hardware that may be available in JSC's water research laboratory (WRL) in the first year. As well, to support the CORBA-DCOM development we will have access to JSC's Environmental System Test Stand (ESTS) that uses OPC hardware (see Facilities and Equipment).

### 1.6.1 Year One: Skills Layer.

Implementing dynamic reconfiguration for the skills layer consists of three efforts: re-implementing the existing skills and the skill level user interface in CORBA, incorporating the CORBA to DCOM bridge for OPC-based devices, and conducting the skills level experiments (in Figure 8 items 1 through 4 and 10 for the UI).

**Re-implementing 3T skills and Skills UI in CORBA.** Using the existing skill sets from the iWRS, we will design IDLs for the devices skills and the intra-skill interfaces, generate the stubs, incorporate new CORBA methods, and test the resulting skills against a skill level simulation previously developed for the iWRS test. We will also recast in CORBA our skill manager graphical user interfaces (GUIs) that display data broadcast by the device skills.

**Building the CORBA to DCOM Bridge.** Based on the OMG COM/CORBA Interworking Specification, a number of commercial vendors have developed bridging tools to support interoperability between DCOM and CORBA (Geraghty et al., 1999). In this effort we will survey the set of existing CORBA to DCOM bridge implementations and select one among them for the basis of our bridge development. We expect the majority of this effort will be spent incorporating the bridging software into our computing and network infrastructure. After the basic implementation has been tested, we will test the bridge with a suite of OPC-based hardware controlling a plant lighting stand in the Environmental System Test Stand (ESTS, see Facilities and Equipment).

**Conducting the Reconfiguration Experiments.** Once the skills level is recast in CORBA, we will run experiments by simulating device failures in an iWRS subsystem for which there is an alternate device (e.g., the PPS) and where no such alternate is available (e.g., the BWP), with both OPC and non-OPC devices. We will also duplicate device skills and build new ones in order to experiment with switching to a new device skill with and without IDL changes. Without a CORBA sequencer we will effect the various reconfigurations by manually starting up and shutting down affected modules. This will give us insights into the reconfiguration algorithms needed for the sequencer in the second year. Our user interface experiments will consist of bringing up and shutting down display and commanding interfaces on a variety of computers in local and remote locations. Finally, we will conduct the same experiments with the OPC hardware on the ESTS. A year-end report will document our findings for dynamically reconfiguring the skills layer of 3T.

### 1.6.2 Year Two: Sequencer Layer.

Implementing dynamic reconfiguration for the sequencer layer consists of two efforts: re-implementing the existing sequencer and sequencer UI in CORBA and conducting experiments that affect the sequencer directly or indirectly (items 1 through 6, 9 and 10 for the sequencer UI in Figure 8).

**Re-implementing the Sequencer and Associated UI in CORBA.** Using the existing 3T sequencer software from the iWRS, we will (1) recast the existing sequencer activation, deactivation, query and event functions used in the skill level as methods defined in the skill IDLs, and (2) define the sequencer IDL and build the methods used by the skills to communicate data to the sequencer. Additionally, the methods used by the sequencer to display data and take commands from the sequencer GUI will be re-implemented as CORBA methods defined in a GUI interface to the sequencer IDL.

**Conduct Sequencer Experiments** In the first part of the year, we will augment the sequencer's existing procedures to include procedures to dynamically detect skill failures and reconfigure the skills level. Next we will rerun experiments 1 through 4 in Figure 8 to test the sequencer's response to changes in the bottom layer. Then, using a duplicate sequencer we will conduct experiments 5 and 6 to test reconfiguration across the bottom two layers. Experiment 9 will be conducted using a simple “expert” application for fault diagnosis that will involve an interface to a user who will conduct the actual analysis and post the information back to the FD module. A

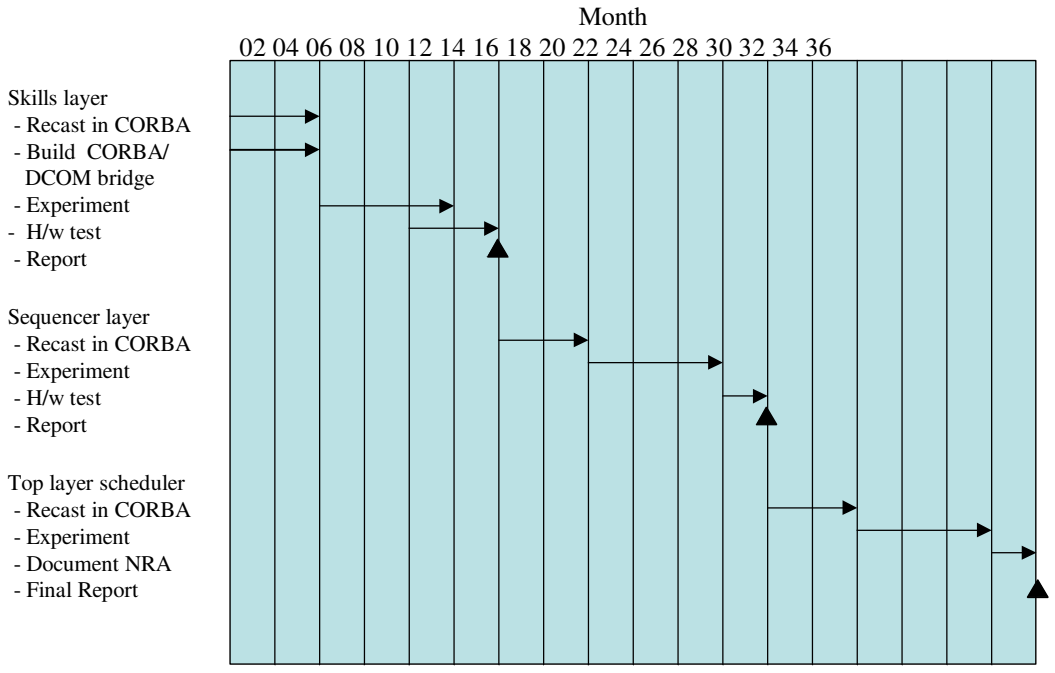
year-end report will document our findings for dynamically reconfiguring the sequencer layer of 3T.

### 1.6.3 Year Three: The Top Layer

As in the previous two years, the effort in this layer will consist of recasting in CORBA the existing interfaces at the top level and then conducting the appropriate experiments (5 through 9 and 10 in Figure 8). During the iWRS test we only had need for a simple planner at the top level to schedule BWP maintenance. As a result of the iWRS test however, the test team determined overall system configurations to support various numbers of crew and various levels of water processing requirements. In year three then, we will develop a long duration scenario involving crew changes and changing water processing requirements in order to generate plans for system reconfiguration. To accomplish this planning we will replace the simple planner with the main 3T planner (Elsaesser and Sanborn, 1990). To this planner must be added operators for both reconfiguring the iWRS for scheduled changes in the scenario, and for dynamically reconfiguring the 3T system to support the single thread experiments for items 5 through 9. To test the full inertia preserving capabilities of a CORBA-based 3T, we will need to recast these experiments against the backdrop of the mission scenario so that we can investigate failure recovery in the middle of carrying out planned reconfigurations of the full iWRS system. Such experiments test the limits of a dynamically reconfigurable 3T.

### 1.6.4 Schedule

A 36-month schedule for accomplishing the above plan is shown below. We have included a report task to document the experience of using CORBA for 3T and comparing its operation with that of the IPC-based version used for the iWRS test.



## 1.7 References


Barber, K. S., Goel, A., Han, D. C., Kim, J., Lam, D. N., Liu, T. H., MacMahon, M. T., McKay, R. M., and Martin, C. E. 2001. Infrastructure for Design, Deployment and Experimentation of Distributed Agent-based Systems: The Requirements, the Technologies, and an Example. In *Proceedings of the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2001)*, 92-99.

Barber, K. S. and Martin, C. E. 2001. Dynamic Adaptive Autonomy in Multi-Agent Systems: Representation and Justification. *International Journal of Pattern Recognition and Artificial Intelligence* 15(3): 405-433.

Barber, K. S., McKay, R. M., Goel, A., Han, D. C., Kim, J., Liu, T.-H., and Martin, C. E. 2000. Sensible Agents: The Distributed Architecture and Testbed. *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems* E83-B(5): 951-960.

Bonasso, R. P. 2001. Intelligent Control of a NASA Advanced Water Recovery System. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey*. Montreal.

Bonasso, R. P., Firby, J. R., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9: 237-256.

Elsaesser, C. and Sanborn, J. 1990. An Architecture for Adversarial Planning. *IEEE Transactions on Systems, Man, and Cybernetics* 20(1): 186-194.

Gat, E. 1998. Three-Layer Architectures. In *Mobile Robots and Artificial Intelligence*, Kortenkamp, D., Bonasso, R. P., and Murphy, R., Eds.: AAAI Press.

Geraghty, R., Joyce, S., Moriarty, T., and Noone, G. 1999. *COM-CORBA Interoperability*. Upper Saddle River, NJ: Prentice Hall PTR.

Henninger, D. 2001. A Vision. In *Briefing on the integrated human exploration mission simulation facility (Integrity)*. Houston: EC, NASA-JSC.

Iconics, I. 2002. Web-enabled OPC Automation at Your Fingertips. <http://www.iconics.com/>.

Lai-fook, K. M. and Ambrose, R. O. 1997. Automation of Bioregenerative Habitats for Space Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2471-2476. Albuquerque, NM: IEEE.

Microsoft. 2002. COM: Delivering on the Promises of Component Technology. <http://www.microsoft.com/com/default.asp>.

National Instruments. 2002. NI Labview. <http://sine.ni.com/apps/we/nioc.vp?cid=1381&lang=US>.

NRC. 1997. *Report of the National Research Council (NRC) Committee on Advanced Technology for Human Support in Space, Aeronautics and Space Engineering (ASEB)*, ISBN 0-309-05744-2, National Research Council, Washington DC.

Office of Biological and Physical Research. 2002. *Advanced Human Support Technology Program*, NRA 02-OBPR-01, NASA, Washington DC.

OMG. 2002. CORBA Basics. <http://www.omg.org/gettingstarted/corbafaq.htm>.

Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M., and Williams, B. 1998. An Autonomous Spacecraft Agent Prototype. *Autonomous Robotics* 5.

Raj, G. S. 1998. A Detailed Comparison of CORBA, DCOM, and Java/RMI. <http://www.execpc.com/~gopalan/misc/compare.html>.

RTI. 2002. Introduction to NDDS. <http://www.rti.com/products/ndds/literature.html>.

Schreckenghost, D., Edeen, M., Bonasso, R. P., and Erickson, J. 1998a. Integrated Control of Product Gas Transfer for Air Revitalization. In *Proceedings of the 28th International Conference on Environmental Systems*. Danvers, MA.

Schreckenghost, D., Martin, C. E., Bonasso, R. P., Kortenkamp, D., Milam, T., and Thronesbery, C. 2002. Supporting Group Interaction Among Humans and Autonomous Agents. In

*Proceedings of the AAAI-02 Workshop on Autonomy, Delegation, and Control: From Inter-agent to Groups*. Edmonton, Canada.

Schreckenghost, D., Ryan, D., Thronesbery, C., Bonasso, R. P., and Poirot, D. 1998b. Intelligent Control of Life Support Systems for Space Habitats. In *Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence*, 1140-1145. Madison, WI: AAAI Press / The MIT Press.

Simmons, R. and Dale, J. 1997. *Inter-Process Communication: A Reference Manual. IPC Version 6.0*: CMU Robotics Institute.

Software AG. 2002. *EntireX DCOM under Linux: Installation and Operation*, Technical Manual. <http://www.softwareag.com/entirex/download/dcomlinux_611.pdf>.

TechnoSoftware. 2002. OPC Solutions. <http://www.technosoftware.com/opcsolutions/pricingusa.html#OPCLinux>.

# 2 Management Approach

Investigators at JSC and at the TRACLabs division of Metrica, Inc., both located in Houston, TX, will carry out the scheduled work for this project in NASA-JSC laboratories (see Facilities and Equipment). As principle investigator, Pete Bonasso will manage the project execution as well as carry out the implementation of CORBA for the top two layers of the architecture. Cheryl Martin will lead the overall design for integrating CORBA into 3T, in particular, the development of the IDLs for each level of the architecture. She will also design and guide the development of the CORBA to DCOM bridge. David Kortenkamp will lead the design and development of the CORBA related code changes to the skill managers. All three investigators have participated in the development of this proposal and are committed to and available for carrying out the research described herein at the levels of effort indicated if funded.

# 3 Personnel Biographical Sketches

## Pete Bonasso

Mr. Bonasso is a principal designer of the 3T intelligent control architecture, and since 1995, has investigated the application of 3T to advanced life support systems, where the key requirement has been the minimization of human vigilance. He has applied 3T to the monitoring and control of plant nutrient delivery systems and an advanced air revitalization system (ARS), and he developed the system planner for managing crew/plant gas exchange during a ninety-one day ALS human test. Since 1998, he has been the designer and developer of the 3T control system supporting the Node 3 advanced ALS technology demonstration, the 450-day biological water processing test and the two-year iWRS test. Mr. Bonasso is a co-founder of the Annual AAAI Robot Competition and Exhibition (Dean & Bonasso 93, Bonasso & Dean 97). He has served on the program committees for the National AI Conference and the annual Agent Theory, Architectures, and Languages conference. He is also an editor of *Artificial Intelligence and Mobile Robots*, with David Kortenkamp and Robin Murphy, published by the MIT Press in 1998. Mr. Bonasso received his B.S. in engineering from the U.S. Military Academy at West Point in 1968 and master's degrees in operation research and computer utilization from Stanford in 1974.

## Mr. Bonasso's Recent Publications


Bonasso, R. P. 2001. Intelligent Control of a NASA Advanced Water Recovery System. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space Odyssey*. Montreal.

Kortenkamp, D., Bonasso, R. P., and Subramanian, D. 2001. Distributed, Autonomous Control of Space Habitats. In *Proceedings of the IEEE Aerospace Conference*.

Kortenkamp, D., Keirn-Schreckenghost, D., and Bonasso, R. P. 2000. Adjustable Control Autonomy for Manned Space Flight. In *Proceedings of the IEEE Aerospace Conference*. Big Sky, MT.

Malin, J., Nieten, J., Schreckenghost, D., MacMahon, M., Graham, J., Thronesbery, C., Bonasso, R. P., Kowing, J., and Fleming, L. 2000. Multi-agent Diagnosis and Control of an Air Revitalization System for Life Support in Space. In *Proceedings of the IEEE Aerospace Conference*.

Schreckenghost, D., Bonasso, R. P., Hudson, M. B., and Kortenkamp, D. 2000. Activity Planning for Long Duration Space Missions. In *Proceedings of the AAAI Workshop on Representational Issues for Real-World Planning Systems*.

Schreckenghost, D., Bonasso, R. P., Kortenkamp, D., and Ryan, D. 1998. Three Tier Architecture for Controlling Space Life Support Systems. In *Proceedings of the IEEE Symposium on*

*Intelligence in Automation and Robotics*: IEEE.
Schreckenghost, D., Edeen, M., Bonasso, R. P., and Erickson, J. 1998. Integrated Control of Product Gas Transfer for Air Revitalization. In *Proceedings of the 28th International Conference on Environmental Systems*. Danvers, MA.
Schreckenghost, D., Martin, C. E., Bonasso, R. P., Kortenkamp, D., Milam, T., and Thronesbery, C. 2002. Supporting Group Interaction Among Humans and Autonomous Agents. In *Proceedings of the AAAI-02 Workshop on Autonomy, Delegation, and Control: From Inter-agent to Groups*. Edmonton, Canada.
Schreckenghost, D., Ryan, D., Thronesbery, C., Bonasso, R. P., and Poirot, D. 1998. Intelligent Control of Life Support Systems for Space Habitats. In *Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence*, 1140-1145. Madison, WI: AAAI Press / The MIT Press.
Schreckenghost, D., Thronesbery, C., Bonasso, R. P., Kortenkamp, D., and Martin, C. E. 2002. Applying Human-Centered Computing to Intelligent Control of Life Support for Space Missions. *IEEE Intelligent Systems*.

## Dr. Cheryl Martin

Cheryl Martin received her B.S. in electrical engineering from Virginia Tech in 1995. She received her M.S. and Ph.D. in computer engineering from the University of Texas in 1997 and 2001 respectively. Her primary area of expertise is software engineering for complex, distributed systems exhibiting artificial intelligence capabilities. Over the past four years, Dr. Martin has acted as a principal designer for two other complex distributed systems based on CORBA: the Sensible Agent Testbed at The University of Texas (Barber and Martin, 2001; Barber et al., 2000) and the Distributed Crew Interaction project at JSC/TRACLabs (Schreckenghost et al., 2002). In conjunction with these projects, Dr. Martin has published several research papers discussing design and required infrastructure for distributed AI systems (Barber et al., 2001; Barber et al., 2000). She has expertise in code development on both Linux and Windows platforms in a variety of programming languages including Java and C++. Dr. Martin's previous electrical engineering experience includes design and development for several real-time control and data-acquisition systems at NASA LaRC for both wind-tunnel instrumentation and robotics applications.

## Dr. Martin's Recent Publications

Barber, K. S., McKay, R. M., Goel, A., Han, D. C., Kim, J., Liu, T.-H., and Martin, C. E. 2000. Sensible Agents: The Distributed Architecture and Testbed. *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems* E83-B(5): 951-960.
Barber, K. S., McKay, R. M., MacMahon, M. T., Martin, C. E., Lam, D. N., Goel, A., Han, D. C., and Kim, J. 2001. Sensible Agents: An Implemented Multi-Agent System and Testbed. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, 92-99. Montreal, QC, Canada.
Barber, K. S. and Martin, C. E. 2001a. Dynamic Adaptive Autonomy in Multi-Agent Systems: Representation and Justification. *International Journal of Pattern Recognition and Artificial Intelligence* 15(3): 405-433.
Barber, K. S. and Martin, C. E. 2001b. Dynamic Reorganization of Decision-Making Groups. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, 513-520. Montreal, QC, Canada.
Barber, K. S. and Martin, C. E. 2001c. The Motivation for Dynamic Decision-Making Frameworks in Multi-Agent Systems. In *Agent Engineering*, Jiming Liu, Zhong, N., Tang, Y. Y., and Wang, P. S. P., Eds. Singapore: World Scientific, 59-91.
Barber, K. S., Martin, C. E., Reed, N. E., and Kortenkamp, D. 2001. Dimensions of Adjustable

Autonomy. In *Advances in Artificial Intelligence: PRICAI 2000 Workshop Reader. Four Workshops held at PRICAI 2000, Melbourne, Australia, August/September 2000. Revised Papers*, vol. LNAI 2112, Kowalczyk, R., Loke, S. W., Reed, N. E., and Williams, G., Eds. Berlin: Springer, 353-361.

Kortenkamp, D., Schreckenghost, D., and Martin, C. E. 2002. User Interaction with Multi-Robot Systems. In *Proceedings of the NATO Multi-Robot Systems Workshop*. Naval Research Laboratory, Washington, D.C.

Martin, C. E. 2001. Adaptive Decision-Making Frameworks for Multi-Agent Systems. PhD Dissertation, Electrical and Computer Engineering, University of Texas at Austin.

Schreckenghost, D., Martin, C. E., Bonasso, R. P., Kortenkamp, D., Milam, T., and Thronesbery, C. 2002. Supporting Group Interaction Among Humans and Autonomous Agents. In *Proceedings of the AAAI-02 Workshop on Autonomy, Delegation, and Control: From Inter-agent to Groups*. Edmonton, Canada.

Schreckenghost, D., Thronesbery, C., Bonasso, R. P., Kortenkamp, D., and Martin, C. E. 2002. Applying Human-Centered Computing to Intelligent Control of Life Support for Space Missions. *IEEE Intelligent Systems*.

## Dr. David Kortenkamp

David Kortenkamp received his B.S. in computer science from the University of Minnesota in 1988. He received his M.S. and Ph.D. in computer science and engineering from the University of Michigan in 1990 and 1993 respectively. A co-designer of the 3T intelligent control architecture, Dr. David Kortenkamp has worked at TRACLabs in support of JSC intelligent robotics programs since 1994. Dr. Kortenkamp has worked on a number of ALS control applications, including the 15-day early human test, the Node 3 advanced ALS technology demonstration, the 450-day biological water processing test, and the iWRS test. Dr. Kortenkamp was chair of the 1999 IJCAI Workshop on Adjustable Autonomy Systems, and served on the program committee for the National Conference on Artificial Intelligence and as a guest editor of a special issue of the Journal of Experimental and Theoretical Artificial Intelligence devoted to intelligent control architectures. He is also an editor of *Artificial Intelligence and Mobile Robots*, with Pete Bonasso and Robin Murphy, published by the MIT Press in 1998.

## Dr. Kortenkamp's Recent Publications

Barber, K. S., Martin, C. E., Reed, N. E., and Kortenkamp, D. 2001. Dimensions of Adjustable Autonomy. In *Advances in Artificial Intelligence: PRICAI 2000 Workshop Reader. Four Workshops held at PRICAI 2000, Melbourne, Australia, August/September 2000. Revised Papers*, vol. LNAI 2112, Kowalczyk, R., Loke, S. W., Reed, N. E., and Williams, G., Eds. Berlin: Springer, 353-361.

Dorais, G. A. and Kortenkamp, D. 2001. Designing Human-Centered Autonomous Agents. In *PRICAI Workshop Reader*, Kowalcyk, R., Lake, S. W., Reed, N., and Willaims, G., Eds. New York: Springer-Verlag.

Kortenkamp, D., Bonasso, R. P., and Subramanian, D. 2001. Distributed, Autonomous Control of Space Habitats. In *Proceedings of the IEEE Aerospace Conference*.

Kortenkamp, D., Keirn-Schreckenghost, D., and Bonasso, R. P. 2000. Adjustable Control Autonomy for Manned Space Flight. In *Proceedings of the IEEE Aerospace Conference*. Big Sky, MT.

Kortenkamp, D., Milam, T., Simmons, R., and Fernandez, J. L. 2001. Collecting and Analyzing Data from Distributed Control Programs. *Electronic Notes in Theoretical Computer Science* 55(2).

Kortenkamp, D., Schreckenghost, D., and Martin, C. E. 2002. User Interaction with Multi-Robot

Systems. In *Proceedings of the NATO Multi-Robot Systems Workshop*. Naval Research Laboratory, Washington, D.C.

Schreckenghost, D., Bonasso, R. P., Hudson, M. B., and Kortenkamp, D. 2000. Activity Planning for Long Duration Space Missions. In *Proceedings of the AAAI Workshop on Representational Issues for Real-World Planning Systems*.

Schreckenghost, D., Bonasso, R. P., Kortenkamp, D., and Ryan, D. 1998. Three Tier Architecture for Controlling Space Life Support Systems. In *Proceedings of the IEEE Symposium on Intelligence in Automation and Robotics*: IEEE.

Schreckenghost, D., Martin, C. E., Bonasso, R. P., Kortenkamp, D., Milam, T., and Thronesbery, C. 2002. Supporting Group Interaction Among Humans and Autonomous Agents. In *Proceedings of the AAAI-02 Workshop on Autonomy, Delegation, and Control: From Inter-agent to Groups*. Edmonton, Canada.

Schreckenghost, D., Thronesbery, C., Bonasso, R. P., Kortenkamp, D., and Martin, C. E. 2002. Applying Human-Centered Computing to Intelligent Control of Life Support for Space Missions. *IEEE Intelligent Systems.*

# 4 Facilities & Equipment

We will develop and test the new CORBA-based 3T and carry out dynamic reconfiguration experiments in the Intelligent Systems Laboratory (ISL) at JSC, and we will carry out follow on hardware testing in two JSC ALS laboratories.

**Intelligent Systems Laboratory**: The Automation, Robotics and Simulation Division (ARSD) of NASA JSC maintains an Intelligent Systems Laboratory (ISL), which will provide computational infrastructure for this project. The ISL contains several dozen networked Unix and Linux workstations that are administered by a full-time support staff. The ISL also contains Windows and Macintosh PCs as well as special purpose computers, such as those running real-time operating systems for control research. Standard software packages are maintained and routinely upgraded. Software development, testing, and integration of software components will take place in the ISL.

With the exception of the items shown in the detailed budget, software required for this project listed below already exists in the ISL:

- Orbix 3, including Iona's DCOM/CORBA "COMet" bridging tools,
- Lisp, Java, and C++ programming languages and development environments for Unix/Linux,
- 3T development tools,
- ILU ORB for CORBA/Lisp development,
- JacORB for CORBA/Java development,
- TAO for CORBA/C++ development.

**Figure 9. A Rack of iWRS Subsystems in the Water Research Laboratory**

**Water Research Laboratory (WRL).** The Crew and Thermal Systems Division (CTSD) of NASA JSC maintain the Water Research Laboratory, which will provide hardware to test the CORBA-based 3T architecture. The PPS of the 2000-2002 iWRS test remains operational along with the controls rack and computers used by 3T to control the iWRS systems. In order to process water in a stand-alone manner, the PPS has its own feed tank and pump. The computers are linked together via LAN; a separate computer provides a gateway to other JSC laboratories, including the ISL.

**Environmental System Test Stand (ESTS).** The Environmental System Test Stand (ESTS) at Johnson Space Center is used as a testbed to prototype various subsystems for the plant growth chamber, called the BPC (Biomass Production Chamber). The ESTS is being used for testing the operating characteristics of different lighting systems and for testing control system hardware and software components. The control system hardware is currently comprised of National Instruments Fieldpoint I/O devices and Sixnet Ethertrak I/O devices connected via Ethernet. Each of these devices has an OPC server running on computers connected to the same Ethernet that reads data from the devices and has the ability to distribute it to any OPC client on the

network. The server for the Windows NT network is a 700 MHz Pentium III computer with 768Mbytes of RAM, running Windows NT 4.0 Server edition. There is also a DIN rail mounted PC from SBS Technologies being used on the network to forward the Fieldpoint and Sixnet data over the network. It is a Pentium III 700 MHz machine with 256 Mbytes RAM running Windows NT 4.0 Workstation edition. A commercial off the shelf software package called Object Automation from Wellspring Solutions is used for data acquisition and control. This package has built-in alarming and redundant data storage capability. It also serves as both an OPC client and server. Object Automation allows for methods to be written in Java, IEC 61131-3 compliant Ladder Logic, or IEC 61131-3 compliant Functional Block diagrams. All components of the application can be distributed across the NT network.

**Figure 10. Environmental System Test Stand**

# 5 Detailed Budget

| | | | Year 1 | | | |
|---|---|---|---|---|---|---|
| **Direct Labor Costs** | **Org** | **Rate** | **MYE** | **Sub-Totals** | **Totals** | **Notes** |
| Pete Bonasso | Metrica | $ 160,000 | 0.20 | $ 32,000 | | Principle Investigator |
| Cheryl Martin | Metrica | $ 160,000 | 0.40 | $ 64,000 | | Co-Investigator |
| Dave Kortenkamp | Metrica | $ 160,000 | 0.40 | $ 64,000 | | Co-Investigator |
| Direct Labor | | | | | $ 160,000 | |
| | | | | | | |
| **Other Direct Costs** | | | | | | |
| **Subcontracts** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Subcontracts | | | | | $ - | |
| | | | | | | |
| **Consultants** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Consultants | | | | | $ - | |
| | | | | | | |
| **Equipment** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| Windows Workstation | | $ 4,500 | 1 | $ 4,500 | | |
| Equipment | | | | | $ 4,500 | |
| | | | | | | |
| | | | | | | |
| **Supplies** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| Windows 2000 C++ & Java Dev | | | | $ 500 | | |
| Supplies | | | | | $ 500 | |
| | | | | | | |
| **Travel** | | **Cost** | **Persons** | **Sub-Totals** | | **Purpose** |
| PI Meeting | | $1,000 | $2 | $2,000 | | Discuss Progress |
| Agents Conf | | $ 1,000 | $ 2 | $ 2,000 | | Present paper |
| Travel | | | | | $ 4,000 | |
| | | | | | | |
| **Other Costs** | | | | **Sub-Totals** | | |
| None | | | | $ - | | |
| Other | | | | | $ - | |
| | | | | | | |
| **Facilities & Administrative Costs** | | | | **Sub-Totals** | | |
| F&A | | | | $ 16,000 | | |
| | | | | | $ 16,000 | |
| | | | | | | |
| **Other Applicable Costs** | | | | **Sub-Totals** | | |
| None | | | | $ - | | |
| Other | | | | | $ - | |
| | | | | | | |
| | | | | | | |
| **Proposal Estimated Costs** | | | | | $ 185,000 | |
| | | | | | | |
| **Proposed Cost Sharing** | | | | | $ - | |
| | | | | | | |
| **Carryover Funds** | | | | | $ - | |
| | | | | | | |
| **Total Estimated Costs** | | | | | $ 185,000 | |

| | | | Year 2 | | | |
|---|---|---|---|---|---|---|
| **Direct Labor Costs** | **Org** | **Rate** | **MYE** | **Sub-Totals** | **Totals** | **Notes** |
| Pete Bonasso | Metrica | $ 168,000 | 0.50 | $ 84,000 | | Principle Investigator |
| Cheryl Martin | Metrica | $ 168,000 | 0.25 | $ 42,000 | | Investigator |
| Dave Kortenkamp | Metrica | $ 168,000 | 0.25 | $ 42,000 | | Investigator |
| Direct Labor | | | | | $ 168,000 | |
| | | | | | | |
| **Other Direct Costs** | | | | | | |
| **Subcontracts** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Subcontracts | | | | | $ - | |
| | | | | | | |
| **Consultants** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Consultants | | | | | $ - | |
| | | | | | | |
| **Equipment** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Equipment | | | | | $ - | |
| | | | | | | |
| **Supplies** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Supplies | | | | | $ - | |
| | | | | | | |
| **Travel** | | **Cost** | **Persons** | **Sub-Totals** | | **Purpose** |
| PI Meeting | | $1,000 | $2 | $2,000 | | Discuss Progress |
| Agents Conf | | $ 1,000 | $ 2 | $ 2,000 | | Present paper |
| Travel | | | | | $ 4,000 | |
| | | | | | | |
| **Other Costs** | | | | **Sub-Totals** | | |
| Other | | | | | $ - | |
| | | | | | | |
| **Facilities & Administrative Costs** | | | | **Sub-Totals** | | |
| F&A | | | | $ 16,800 | | |
| | | | | | $ 16,800 | |
| **Other Applicable Costs** | | | | **Sub-Totals** | | |
| None | | | | $ - | | |
| Other | | | | | $ - | |
| | | | | | | |
| | | | | | | |
| **Proposal Estimated Costs** | | | | | $ 188,800 | |
| | | | | | | |
| **Proposed Cost Sharing** | | | | | $ - | |
| | | | | | | |
| **Carryover Funds** | | | | | $ - | |
| | | | | | | |
| **Total Estimated Costs** | | | | | $ 188,800 | |

| | | | Year 3 | | | |
|---|---|---|---|---|---|---|
| **Direct Labor Costs** | **Org** | **Rate** | **MYE** | **Sub-Totals** | **Totals** | **Notes** |
| Pete Bonasso | Metrica | $ 176,000 | 0.50 | $ 88,000 | | Investigator |
| Cheryl Martin | Metrica | $ 176,000 | 0.40 | $ 70,400 | | Investigator |
| Dave Kortenkamp | Metrica | $ 176,000 | 0.10 | $ 17,600 | | Investigator |
| Direct Labor | | | | | $ 176,000 | |
| | | | | | | |
| **Other Direct Costs** | | | | | | |
| **Subcontracts** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Subcontracts | | | | | $ - | |
| | | | | | | |
| **Consultants** | **Org** | **Rate** | **Hours** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Consultants | | | | | | |
| | | | | | | |
| **Equipment** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Equipment | | | | | $ - | |
| | | | | | | |
| **Supplies** | | **Unit Cost** | **Qty** | **Sub-Totals** | | |
| None | | | | $ - | | |
| Supplies | | | | | $ - | |
| | | | | | | |
| **Travel** | | **Cost** | **Persons** | **Sub-Totals** | | **Purpose** |
| PI Meeting | | $1,000 | $2 | $2,000 | | Discuss Progress |
| Agents Conf | | $ 1,000 | $ 2 | $ 2,000 | | Present paper |
| Travel | | | | | $ 4,000 | |
| | | | | | | |
| **Other Costs** | | | | **Sub-Totals** | | |
| none | | | | $ - | | |
| Other | | | | | $ - | |
| | | | | | | |
| **Facilities & Administrative Costs** | | | | **Sub-Totals** | | |
| F&A | | | | $ 17,600 | | |
| | | | | | $ 17,600 | |
| **Other Applicable Costs** | | | | **Sub-Totals** | | |
| None | | | | $ - | | |
| Other | | | | | $ - | |
| | | | | | | |
| **Proposal Estimated Costs** | | | | | $ 197,600 | |
| | | | | | | |
| **Proposed Cost Sharing** | | | | | $ - | |
| | | | | | | |
| **Carryover Funds** | | | | | $ - | |
| | | | | | | |
| **Total Estimated Costs** | | | | | $ 197,600 | |

# 6 Supporting Budgetary Information

## 6.1 Direct Costs.

Labor rates are the latest government approved rates for personnel at Metrica, Inc. In the first year we will also need $5000 for hardware and software purchases as follows:

- A Windows/Linux laptop to be used in the ESTS for testing the DCOM-CORBA bridge ($4500)
- Windows 2000 software development environment for C++ and Java ($500),

The remainder of the software required for this project listed below already exists in the ISL and ESTS facilities (see Facilities & Equipment)

- Orbix 3, including Iona's DCOM/CORBA "COMet" bridging tools,
- Lisp, Java, and C++ programming languages and development environments for Unix/Linux,
- 3T development tools,
- OPC development environments and servers,
- ILU ORB for CORBA/Lisp development,
- JacORB for CORBA/Java development,
- TAO for CORBA/C++ development.

## 6.2 Indirect Costs.

F&A costs consist of the current NASA-JSC rate of 10% of direct labor costs